Adapting the RACER Architecture to Integrate Improved In-ReRAM Logic Primitives

Minh S. Q. Truong*, Liting Shen*, Alexander Glass*, Alison Hoffmann*
L. Richard Carley*, James A. Bain*, Saugata Ghose†
*Carnegie Mellon University, †University of Illinois Urbana-Champaign

***Abstract*—Modern computing applications based upon machine learning can incur significant data movement overheads in state-of-the-art computers. Resistive-memory-based *processing-using-memory* (PUM) can mitigate this data movement by instead performing computation *in situ* (i.e., directly within memory cells), but device-level limitations restrict the practicality and/or performance of many PUM architecture proposals. The RACER architecture overcomes these limitations, by proposing efficient peripheral circuitry and the concept of bit-pipelining to enable high-performance, high-efficiency computation using small memory tiles. In this work, we extend RACER to adapt easily to different PUM logic families, by (1) modifying the device access circuitry to support a wide range of logic families, (2) evaluating three logic families proposed by prior work, and (3) proposing and evaluating a new logic family called OSCAR that significantly relaxes the switching voltage constraints required to perform logic with resistive memory devices. We show that the modified RACER architecture, using the OSCAR logic family, can enable practical PUM on real ReRAM devices while improving performance and energy savings by 30% and 37%, respectively, over the original RACER work.**

I. Introduction

The rise of machine learning (ML) has led to the development of many modern data-intensive computing applications (e.g., image classification, object tracking, robotic navigation, text prediction) that perform ML algorithms as a key component of the larger application. Unfortunately, data movement between the CPU and memory components can consume as much as two orders of magnitude more energy than that needed to process the data [14, 32], and can be responsible for a majority of energy consumed by data-intensive applications [10]. This is particularly costly in mobile and edge computing platforms, where energy is a first-order design concern. For data-centric applications, to maximize energy efficiency, these platforms ideally should avoid *both* sending large amounts of data across the network and inducing large amounts of data movement during local/edge processing.

To mitigate data movement overheads, recent works have proposed new device innovations based on the principle of *processing-using-memory* (PUM) [22, 49]. The PUM paradigm takes advantage of electrical interactions between interconnected memory cells to perform primitive computational functions, in addition to the original role of the cells as data storage. Examples of these primitives include various families of Boolean-complete operators (e.g., [23, 30, 33, 48]) and multi-bit dot products (e.g., [5, 13, 50, 51]). Such primitives can be performed *in situ* on the data (i.e., the data never has to leave the memory cell). The principles of PUM have been demonstrated using a wide range of memory technologies, including more conventional DRAM [21, 24, 39, 48] and SRAM [1, 16, 19, 30], and emerging technologies such as resistive memories [2, 3, 4, 5, 13, 20, 23, 25, 26, 28, 33, 34, 35, 37, 40, 50, 51, 57, 59, 62].[1] Resistive memories are attractive alternatives to DRAM as PUM-enabling technologies because of their ability to perform logically-complete bitwise Boolean operations *without* relying on additional compute circuitry, and because of their higher memory densities.

In our previous work, we tackled critical device- and circuit-level limitations that constrain resistive memory array scaling by designing *RACER* (Resistive Accelerated Computation for Energy Reduction) [52]. While large memory arrays have traditionally been seen as an effective way to amortize both peripheral circuit area and high PUM latencies, there is a fundamental limit to increasing the array size in resistive crossbar arrays (see Section II-D). RACER is the first PUM architecture that addresses this array size limitation, relying on a novel execution model that we call *bit-pipelining* to provide high throughput for many important arithmetic operations, which enable the architecture to perform both essential ML computations *and* a large variety of non-ML functions. Even when implemented with very small (i.e., 64×64) arrays based on the MAGIC logic family for ReRAM [33], RACER provides 107× and 12× the performance of a 16-core CPU and a 2304-shader-core GPU, respectively, with energy savings of 189× and 17×, for many data-intensive microbenchmarks [52]. Unfortunately, while these performance and energy improvements are attractive, the use of MAGIC as the underlying logic family places fairly stringent restrictions on the switching voltages of the ReRAM cells.

In this work, we explore the impact of alternate resistive memory technologies on RACER. Specifically, we explore how RACER can be integrated with other previously-proposed logic families (NAND using MAGIC [33], and the multi-primitive FELIX [23]), and we develop a new ReRAM-based logic family approach called *OSCAR* (Optimized Switching Constraints for RACER), which dramatically reduces the constraints on ReRAM switching voltages compared to previously-proposed logic families. OSCAR supports NOR and OR Boolean logic primitives, and can make use of either

[1] In this work, we use the term *resistive memory* to refer broadly to resistance-based non-volatile memories (e.g., PCM, MRAM, ReRAM), while we use ReRAM to refer specifically to oxide-based switches (often referred to as memristors).

1S1R (one selector, one resistor) or 1T1R (one transistor, one resistor) ReRAM devices. We provide a detailed discussion on the circuit-level changes (predominantly in the decode & drive circuits used to assert different voltages on ReRAM array wordlines) needed to integrate these and other logic families into RACER.

We evaluate the modified RACER architecture using four ReRAM-based logic families: (1) NOR using MAGIC [33], our baseline family that was employed in the original RACER work [52]; (2) NAND using MAGIC; (3) NAND, AND, NOR, and XOR using FELIX [23]; and (4) NOR and OR using OSCAR. We find that, ignoring the practicality of switching voltage constraints, the FELIX logic family provides an average performance improvement of 70% over the MAGIC NOR baseline, along with energy savings of 43%, across a range of data-intensive microbenchmarks. Our OSCAR logic family, with its more practical switching voltage constraints that can work with real ReRAM devices, achieves an average performance improvement of 30% and energy savings of 37% over MAGIC NOR.

We make the following contributions in this work:

- We propose a new ReRAM-based logic family called OSCAR. OSCAR can support two logic primitives: NOR and OR. Compared to prior work on ReRAM-based logic, the device constraints for OSCAR are more practically attainable in terms of the ratio of the set and reset voltages. Unlike previously-proposed logic families, this property allows OSCAR to be widely applicable across most proposed ReRAM devices.
- We demonstrate the necessary circuit-level changes to adapt RACER to a wide range of logic families. These changes are generalized based on the shared characteristics of different resistive memory technologies, allowing RACER to integrate with future technologies and logic families without major redesigns.
- We evaluate the performance and energy consumption of different logic families on our modified RACER architecture. We provide detailed estimates of the power and area overheads of RACER across these different logic families.

The rest of the paper is organized as follows: Section II provides background on PUM; Section III motivates the need for better device switching thresholds and the need to extend RACER to integrate with several different logic families; Section IV explains the OSCAR logic family; Section V describes the changes necessary to integrate different logic families with RACER; Section VI discusses our methodology; and Section VII describes our evaluation.

II. Background

In this section, we provide a brief discussion on the underlying technologies that we use to build RACER, along with some of the foundations that motivate our new logical primitive approach (OSCAR) and motivate the need to integrate different resistive memory technologies with RACER.

A. Access Topologies for Resistive Memories

As DRAM scaling issues continue to be difficult to solve [31, 45, 46], researchers have been developing a number of emerging memory alternatives, which are starting to reach commercial deployment [11, 44, 60]. These alternatives include resistive memories such as MRAM, PCM, and ReRAM. One way to implement these technologies is similar to the access topology used for DRAM. Figure 1a shows a typical 1T1C (one transistor, one capacitor) access topology for DRAM [54], while Figure 1b shows a similar 1T1R (one transistor, one resistor) array topology for a resistive memory array [41]. In such an array, the access transistors are fabricated in the *front-end-of-line* (FEOL) process, while the resistor elements are fabricated amid the metallization layers in the *back-end-of-line* (BEOL) process.

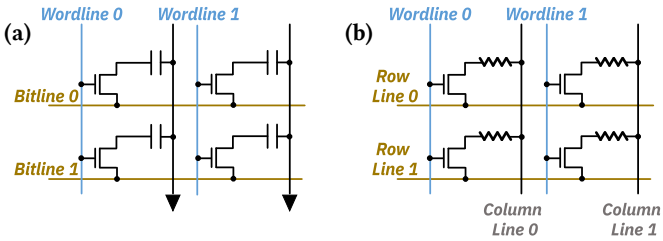


Fig. 1: (a) 1T1C DRAM array; (b) 1T1R resistive memory array.

To eliminate the dense access transistor array in 1T1R topologies, device and circuit researchers have envisioned a way to integrate these emerging memory technologies fully into the BEOL process, by using a *crossbar* array topology and *resistive selectors* for access control [60], as shown in Figure 2a. One resistive selector and one resistor (1S1R) in series sit at the intersection of each row and each column, forming a memory cell. The resistive selector element controls access to the cell, eliminating the need for the separate column line present in the 1T1R topology, and the resistor stores data. Figure 2 shows a typical non-linear, bipolar, two-terminal memory cell for 1S1R, with the resistive selector represented as opposing diodes (as the selector is designed to prevent sneak path currents). Because of this topology's symmetry, we can access data in both the column and row directions, and the access points for 1S1R are named column lines and row lines, correspondingly. In this topology, a single cell can be selected by asserting predetermined selection voltages on one row line and on one column line.

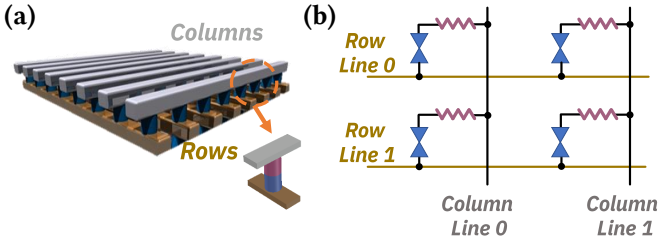


Fig. 2: (a) 1S1R crossbar access topology for resistive memory, with resistive selectors in blue and resistors purple; (b) circuit diagram for a 1S1R memory cell.

B. ReRAM Devices

The redox-based RAM (ReRAM) cell is a non-volatile resistive memory device that stores data in the form of a resistance

value, which can be changed by applying a switching voltage that triggers a redox reaction [55]. Importantly, these devices are bipolar, allowing them to be *set* to a low resistance (logic 1) with a positive pulse and *reset* to a high resistance (logic 0) with a negative pulse. Fundamentally, these elements are compatible with both 1T1R and 1S1R access topologies.

Figure 3a shows an idealized *I-V* curve of a ReRAM cell where the V_{set} and V_{reset} voltages alter the cell's resistance between two resistance levels.[2] Figure 3b shows the *I-V* of a typical TaO_x resistive element, including a negative differential resistance (NDR) region where the differential resistance *dV/dI* is negative. Section IV discusses the importance of this NDR region to realize OSCAR's logic NOR primitive.

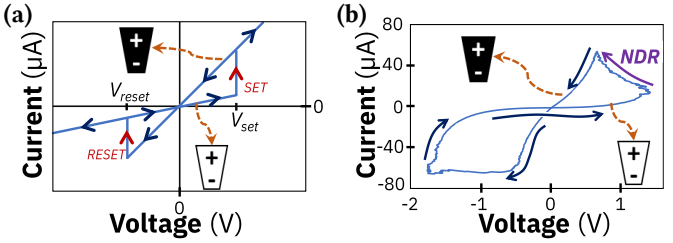


Fig. 3: (a) Idealized ReRAM I-V curve; (b) I-V curve of a real TaO_x resistive element, with negative differential resistance (NDR) region shown as purple arrow.

Some prior work proposes to store multiple bits of data per cell [36], which can enable analog multi-bit in-memory operations such as dot products. However, multi-bit storage requires relatively large resistance ranges in order to divide the range up among the different multi-bit values. In contrast, we treat ReRAM cells as digital devices that each store only a single bit of data. This provides greater immunity to variation in switching thresholds and resistance drift, and avoids the need for complex analog-to-digital converters (ADCs) during processing.

C. Logic Families Using Resistive Memories

There are several proposals to implement logic families (consisting of one or more Boolean logic primitives) through the direct interaction of resistive memory cells. The MAGIC logic family [33] eliminates the load resistors required by previous work [9] and enables resistive memory to perform the NOR logic primitive in a two-step process (output initialization and voltage assertion). Figure 4a summarizes how MAGIC performs NOR using three resistive cells along the same row. Two cells are selected as the inputs to the NOR primitive by asserting a voltage V_{nor} on the cells' corresponding column lines. At the same time, a cell is selected as the output by grounding its column line, and the shared row line is allowed to float at V_{float}. This allows an induced current to flow from the input cells to the output cell, initiating the output cell switch procedure. This NOR primitive is *non-destructive*, i.e., it does not overwrite either of the input cell values.

[2]To illustrate the non-volatile and polarity-dependent switching of ReRAM devices throughout this paper, we use a trapezoidal-shaped element to represent the device. Application of a positive voltage to the wide end of the trapezoid sets the device (to a low resistance), while application of a positive voltage to the narrow end of the device resets the device (to a high resistance). We use a black element to indicate the set (low resistance/logic 1) state, while a white element indicates the reset (high resistance/logic 0) state.

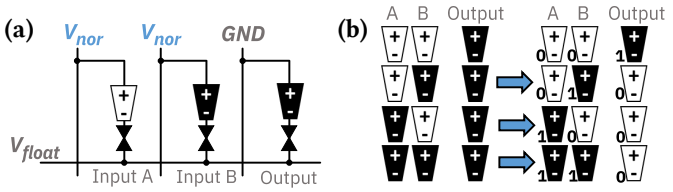


Fig. 4: (a) Voltage assertions for MAGIC's NOR primitive on 1S1R crossbar; (b) possible output transitions for NOR, with arrows indicating output cell resistance switches.

Figure 4b illustrates the possible cell switch cases for the MAGIC NOR primitive. Note that the output cell must be pre-initialized to the low resistance state (logic 1) prior to the application of the V_{nor} voltage to the input columns. An important assumption for resistive-memory-based PUM primitives is that memory state can be set in a single voltage assertion, without closed-loop feedback. This is reasonable given the large resistance ratios and single-bit states of these implementations, and has been the model from the inception of these approaches [9].

The FELIX logic family [23] builds upon the MAGIC voltage assertion concept to provide three primitives (NAND, NOR, OR) that can each be executed in a single clock cycle. The availability of multiple primitives can reduce the total execution time of an application in a PUM architecture by allowing for logic simplification.

D. RACER Architecture

In our previous work [52], we showed that there is a fundamental limit to how large a crossbar can be for PUM architectures that perform *whole-column* operations, in which logic primitives (as discussed in Section II-C) are applied to an entire column of the crossbar. For an $n \times n$ crossbar of ReRAM cells, $n \leq 200$ in such architectures because of the current carrying limit of metal wires. The array size constraint significantly limits the throughput of existing PUM architectures. To work around this constraint, we designed RACER [52], a PUM architecture based on a novel *bit-pipelining execution model* and the MAGIC logic family [33] to achieve high performance while using small *tiles* (i.e., 64×64 crossbars.

In this work, we summarize two features of RACER that are relevant to our discussion: (1) the bit-pipelining execution model; and (2) the RACER cluster, which consists of control and peripheral logic to enable bit pipelining, as well as read/write (R/W) circuitry. In Section V, we discuss how to modify the cluster's control and R/W circuitry to integrate RACER with resistive memory technologies beyond ReRAM, and logic families beyond MAGIC. We refer readers to our MICRO 2021 paper [52] for the full architecture-level details of RACER.

Bit-Pipelining. RACER takes advantage of a novel execution model that we call *bit-pipelining* to concurrently operate on $w \times n$ words, for a w-bit word and an $n \times n$ tile. RACER exploits

parallelism across multiple tiles by striping each bit of a w-bit word across w tiles, as shown in Figure 5 (Tile 0 holds the least significant bit, or LSB). RACER can realize bit-serial operations [7] by iteratively applying column-wide Boolean logic primitives one tile at a time. As an example, RACER performs ripple-carry addition tile-by-tile: at each bit i, it generates the sum and carry-out bits in Tile i for 64 different addition calculations simultaneously, and then propagates the carry-out bits to Tile $i+1$ to compute the next bit.

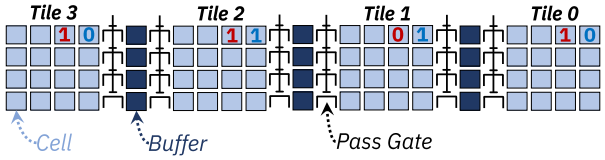


Fig. 5: Tile and buffer design, showing two four-bit values (1101 in red, 0110 in blue) striped across the tiles.

To enable inter-tile data transfer, RACER uses *buffers*, which are 1×64 crossbars (made of the same device as the tiles) that connect to a pair of tiles using programmable pass gates (Figure 5). For our addition example, after computing the carry-out bits in Tile i, RACER connects Tile i to Buffer i, copies the carry-out column into the buffer, and then connects the buffer to Tile $i+1$. The buffer's contents are finally copied into Tile $i+1$, where the column is used as the carry-in bits for bit $i+1$'s addition.

Our decoupled tile-by-tile computation allows us to pipeline *across* bits in RACER. We group w tiles and their corresponding buffers together to form a *pipeline* (we set $w = 64$ to support up to 64-bit computation). After Tile i finishes its computation for the current column (i.e., for the current n words) and passes its generated column(s) to Tile $i+1$, Tile i is free to perform another sequence of combinational logic primitives, on another column with n *different* words of data. This allows RACER to have up to w instructions in flight simultaneously, each operating on n words at once.

RACER Cluster. In RACER, we exploit our observation that many bit-serial operations perform the same Boolean logic primitives on each bit, in order to design efficient control logic. For such operations, RACER generates a sequence of primitives for one tile, and then propagates the sequence from tile to tile. A *byte group* (Figure 6a) contains the control circuitry to enable bit-pipelining across eight consecutive tiles, with one *micro-op queue* per tile. Each queue holds a sequence of *micro-ops*, which are commands that tell RACER which primitive to apply to a tile, and can control which columns are operated on (and when to connect/disconnect adjacent buffers). RACER propagates micro-ops from the head of one micro-op queue to the tail of the next micro-op queue in a byte group. RACER can configure whether adjacent byte groups are connected together to allow propagation across byte groups, which we use to enable bit-pipelining at the 8-/16-/32-/64-bit granularity. The micro-ops drive selection voltages using per-tile *decode & drive* units, which serve as the interface between the technology-agnostic byte group circuits and the technology-specific crossbar voltages.

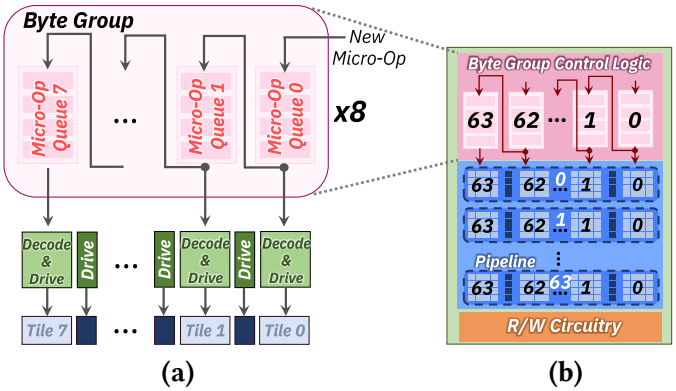


Fig. 6: (a) Byte group; (b) RACER cluster: 8 byte groups, 64 pipelines, and R/W circuitry.

We group 64 pipelines into a *cluster* (Figure 6b), with the pipelines in a cluster receiving commands from a single shared byte group control unit. A RACER chip can contain an arbitrary number of clusters depending on platform needs. Each cluster has its own control units and peripheral circuitry, and can operate independently of other clusters. The cluster also contains read/write (R/W) circuitry, which can read/write data to 64 buffers of the same pipeline in bursts. The R/W circuitry is used to transfer data both between pipelines in the same cluster and between clusters.

III. Motivation

A. Device Motivation: Improved Switching Thresholds

A critical requirement for the logic families described in Section II-C to function correctly is that the reset voltage V_{reset} *must be* less than half of the set voltage V_{set}. MAGIC [33] derives this relationship in detail and shows the consequences of not meeting this criteria (failure to accomplish the logical primitive, or accidental overwrite of the inputs). FELIX requires the same relationship between V_{set} and V_{reset} as MAGIC.

The switching voltage constraints imposed by a logic family are an extremely important consideration for the viability of PUM, as it is often difficult to manipulate the switching voltages of the ReRAM cells to achieve this required relationship (as we discuss in Section IV-C). To this end, we propose a new logical family, OSCAR, that retains the spirit of prior works but attempts to relax the constraints on the switching thresholds and the required load resistors to more practical values. For practical implementation, we believe it is critical to *not* constrain the switching voltages in the ways in which the MAGIC approach does (i.e., we do not want to require that $V_{reset} < V_{set}/2$).

B. Architectural Motivation: Adaptability

As RACER's bit-pipelining execution model addresses the array size limitation that directly affects the architecture's performance, it is highly beneficial for RACER to work with many different resistive memory technologies and logic families. The limited current carrying capacity of metal wires, regardless of topologies and logic primitives, is fundamental to

the wire material, and is expected to hold true for *any* resistive technology capable of performing whole-column operations, not just for MAGIC-capable ReRAM.

To this end, we modify and generalize RACER's decode & drive units and read/write (R/W) circuitry to ensure that RACER can be integrated with new resistive memory technologies, benefiting from technological improvements without the need for major architectural redesigns. RACER was designed to explicitly avoid exposing the programmer to technology-specific capabilities such as the available logic primitives, by providing a general-purpose instruction set architecture (ISA) and a programming abstraction for the bit-pipelining execution model. With the generalized decode & drive units and R/W circuitry acting as the interface to specific technology domains, this allows for the development of PUM-specific, technology-agnostic software, which is a key need to enable the adoption of PUM-based computing [22].

IV. OSCAR: Optimizing Switching Constraints

Given the difficulties that practical ReRAM devices have in meeting the switching voltage constraints ($V_{reset} < V_{set}/2$) required for MAGIC and FELIX, we propose an alternative ReRAM logic family, which we name *OSCAR* (Optimized Switching Constraint Approach to RACER).[3] As we describe below, OSCAR enables two logic primitives: (1) non-destructive NOR and (2) destructive OR (i.e., one of the inputs is overwritten).

A. Non-Destructive NOR

Figure 7a shows how four resistive cells can be used to perform the NOR primitive in OSCAR. Aside from having two input cells and one output cell (just as MAGIC and FELIX do), OSCAR employs a fourth *load* cell to correctly balance the voltage division among the resistive cells. The load resistor is used in a similar manner by the imply-based (IMP) approach [9, 35]. However, unlike the IMP approach (which requires a separate intermediate resistance value), the value of the load resistor in OSCAR can be anything between R_{on} (logic 1) and R_{off} (logic 0). Thus, with OSCAR, one column in a tile can be dedicated to serve as a load without requiring repeated initialization, and the value of the load resistor can be selected to minimize energy (by setting it to R_{off}) or to minimize the voltage applied to the cells (by setting it to R_{on}). In our analysis, we use load resistors that are fixed to logic 0 to minimize energy.

In OSCAR, the output cell is initialized to logic 0 (instead of logic 1 for MAGIC), and the voltage $V_{nor} + \Delta$ is applied to the cell, while the load cell is grounded. Theoretically, the output cell can be asserted using V_{nor}, which is what we assert on the input cells (i.e., without the additional small voltage Δ), but this introduces the risk of destroying the input cells' data. In our analysis below, we first present OSCAR's logic NOR primitive with the output cell asserted using V_{nor}, then we explain how adding Δ helps preserve the data in the input cells.

[3] Apologies to Jack Lemmon, Tony Randall, Walter Matthau, and Jack Klugman, the original movie/television Felixes and Oscars, respectively.

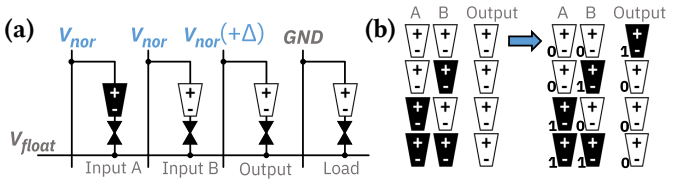


Fig. 7: (a) Voltage assertions for OSCAR's NOR primitive on 1S1R crossbar; (b) possible output transitions for NOR, with arrows indicating output cell resistance switches.

If at least one of the inputs is logic 1, there is a minimal voltage drop on the input and output cells because $R_{on} << R_{off}$, leaving the output unchanged. As shown in Figure 7, the only case where the output changes is when both of the inputs are logic 0 (R_{off}). In this case, the voltage drop on the input and output cells is $V_{nor}/4$, assuming that the input and output cells are all asserted with V_{nor}. If $V_{nor}/4 > V_{set}$, the output changes to logic 1 (i.e., NOR(0,0) = 1). Thus, in OSCAR, the resistance constraint is $R_{on} << R_{off}$ and the voltage constraint is $V_{nor} > 4V_{set}$, while there is *no voltage constraint* between V_{set} and V_{reset} unlike previous works. Note that because the same voltage V_{nor} is applied to the input and output cells, the voltage drop $V_{nor}/4$ exists not only at the output but also at the inputs for the NOR(0,0) case. Thus, the data in the input cells can potentially be overwritten during NOR.

To make OSCAR's NOR non-destructive (i.e., input data is not overwritten after the operation), we increase the voltage asserted on the output cell by Δ. Asserting $V_{nor} + \Delta$ allows the output cell to reach V_{set} before the input cells, such that only the output switches from logic 0 to logic 1. As the output switch happens, the output cell enters a negative differential resistance (NDR) region, where the voltage across the cell decreases but the current flowing through the cell increases ($dV/dI < 0$) due to the rapid decrease of the cell's resistance (see Figure 3b). With a sufficiently large Δ, the maximum voltage for the input cells never reaches V_{set}, preventing the input cells from switching and losing their data. When one of the NOR inputs is logic 1, the maximum voltage drop is less than Δ, so no state change occurs as long as Δ is less than V_{nor}.

Simulation. We have simulated the electrical events that occur during the OSCAR NOR primitive using SPICE. We apply assertion voltage pulses that are 12 ns in length. V_{set} and V_{reset} are both set to 2 V, and Δ is set to 1.5 V. From empirical analysis, setting Δ to approximately 50–75% of V_{set} ensures correct NOR functionality for OSCAR. A smaller value of Δ may still permit the input cells to switch, while a larger value of Δ can overwrite the output cell. Figure 8 shows the current and voltage amplitudes over time during the NOR primitive, as observed for each of the input cells (A and B) and for the output cell. The assertion voltage pulses are applied at $t = 3$ ns. The graphs for NOR(0,1) are similar to those for NOR(1,0), with only *Input A* switched with *Input B*, so we omit NOR(0,1) graphs for brevity.

We first examine the current and voltage changes during NOR(0,0), as shown in Figure 8. Because the output cell is asserted with $V_{nor} + \Delta$ while the input cells are asserted with

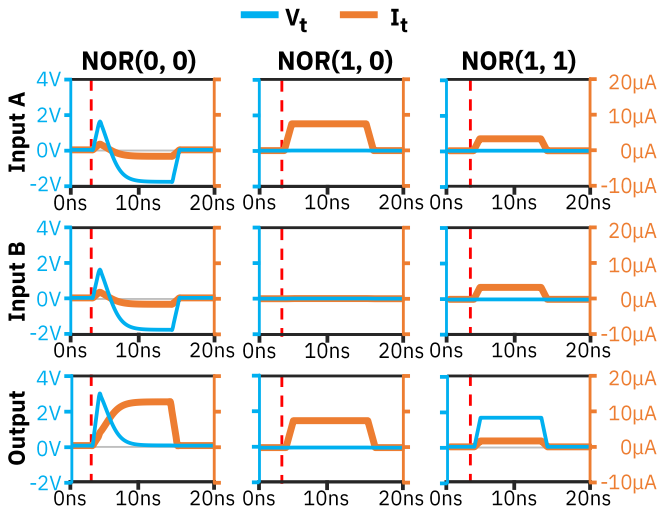


Fig. 8: SPICE results for OSCAR's non-destructive NOR. Red dashed line shows assertion voltage pulse start time.

V_{nor}, the output cell's voltage grows faster than the input cells' voltages. Thus, the output cell enters the NDR region at around $t = 5$ ns. At the same time, the input cells' voltages start to fall, becoming negative at around $t = 5.6$ ns. In the NDR region ($dV/dI < 0$), the output cell's value switches from logic 0 to logic 1 as the device voltage drops rapidly. Meanwhile, the decreasing voltage across the output cell, and by extension the decreasing voltages across the input cells, prevent the inputs from switching ($dV < 0$).

We now examine the current and voltage changes during NOR(1,0) and NOR(1,1), as shown in Figure 8. To understand the changes, we can apply the voltage superposition principle and divide the voltages asserted on the input and output cells into two parts: (1) V_{nor}, V_{nor}, V_{nor} to *Input A*, *Input B*, and *Output*, respectively; and (2) 0, 0, Δ to *Input A*, *Input B*, and *Output*, respectively. For the first part, because at least one of the input cells is set to logic 1, there is virtually no voltage drop across the input and output cells. For the second part, an additional voltage Δ is applied to the output cell. Thus, the voltage drop across the output cell is Δ, as we observe in the figure. From this analysis, we show that as long as Δ is sufficiently lower than V_{set}, the output cell does not switch when at least one of the input cells is set to logic 1.

B. Destructive OR

To expand OSCAR's capabilities, we introduce a destructive OR logic primitive. This primitive, which does not require the output to be pre-initialized, offers improved efficiency when the input operands do not need to be preserved.

Figure 9a shows the two ReRAM cells involved in the destructive OR operation. Unlike OSCAR's NOR primitive, OR requires only two input cells (one of which will be overwritten), and the two inputs are asserted using two different voltages (i.e., V_{or} and ground). The input cell asserted using V_{or} is overwritten with the output value at the end of the operation (i.e., *Input A* in Figure 9a).

V_{or} must be selected carefully in order for OSCAR to perform destructive OR. We identify three constraints on V_{or}

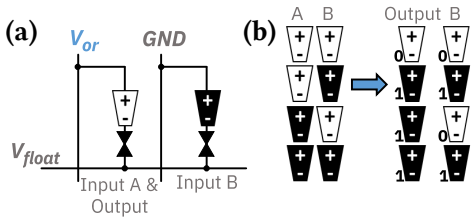


Fig. 9: (a) Voltage assertions for OSCAR's OR primitive on 1S1R crossbar; (b) possible cell transitions for OR, with arrows indicating cell resistance switches.

for different input cases, which need to be satisfied for the OR operation to work. In the cases of OR(0,0) = 0 and OR(1,1) = 1, we do not want the cells to switch because *Input A* already has the correct result. This can be achieved if $V_{or}/2 < V_{set}$ (❶). The voltage drops across the two cells are equal in this case, since the cells' resistances are the same. However, to prevent *Input B* from switching, we need $V_{reset} > V_{or}/2$ (❷). In the case of OR(0,1) = 1, where *Input A* is set to logic 0, most of the voltage drops across *Input A*, allowing the cell to switch. For this case, we need $V_{or} > V_{set}$ (❸). In the case of OR(1,0) = 1, where *Input A* is set to logic 1, most of the voltage drops over *Input B*. However, because of *Input B*'s polarity, the cell does not switch. As a result, no constraints on V_{or} are required for this case. By combining the three constraints (❶, ❷, ❸), we obtain the constraints $V_{set} < V_{or} < min(2V_{set}, 2V_{reset})$, which can be simplified to $V_{set} < V_{or} < 2V_{reset}$.

Simulation. Figure 10 shows the current and voltage changes observed when we use SPICE to simulate the OR primitive. The assertion voltage pulses are applied at $t = 3$ ns. We choose V_{or} to be 3 V, which satisfies the constraints for OSCAR OR. For OR(0,0) and OR(1,1), we observe equal voltages drop over both cells, although for OR(0,0) we observe that a minimal current runs through the cells due to their high total resistance. For OR(1,0), we observe that the voltage drop across *Input B* is greater than 2 V, but the cell does not switch due to its polarity. For OR(0,1), we observe *Input A* going through the NDR region when it switches.

C. Relaxed Constraints and Process Variation

MAGIC [33], FELIX [23], and OSCAR all impose certain switching constraints on the device in order to construct useful logic primitives. We observe that these constraints typically involve two types of variables: (1) technology-related variables (i.e., V_{set}, V_{reset}); and (2) logic-related variables (V_{logic}; e.g., V_{nor}, V_{or}). To demonstrate OSCAR's ability to relax the switching constraints, we introduce two quantities that correlate these variables: V_{logic}/V_{reset} and V_{logic}/V_{set}. All of the switching constraints for MAGIC, FELIX, and OSCAR can be expressed using these two quantities. As an example, MAGIC and FELIX require that $2V_{reset} < V_{nor} < V_{set}$, which can be expressed as $V_{logic}/V_{reset} > 2$ and $V_{logic}/V_{set} < 1$. Figure 11 plots the constraints for MAGIC/FELIX, OSCAR NOR, and OSCAR OR as colored regions. The figure also shows the constraint region of typical (real) resistive devices

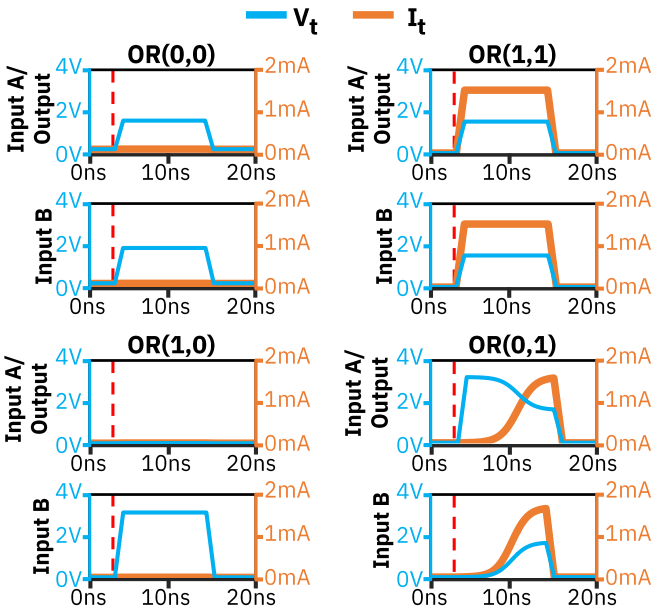


Fig. 10: SPICE results for OSCAR's destructive OR. *Input A* also serves as *Output*. Red dashed line shows assertion voltage pulse start time.

(shown as a hashed blue region), which usually have $V_{set} < 2V_{reset}$ [12, 18, 27, 36, 43, 56, 58, 61]. We see that OSCAR NOR and OSCAR OR overlap with the typical device region, while MAGIC/FELIX do not. We conclude that OSCAR relaxes the switching constraints and makes it easier for typical real devices to perform logic primitives.

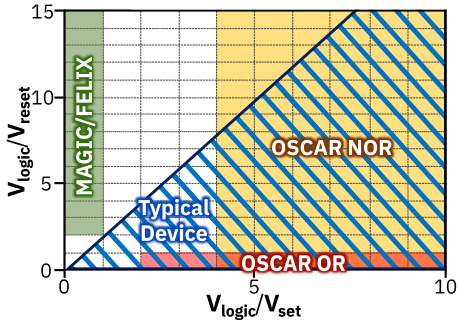


Fig. 11: Threshold voltage constraints of a typical real device (blue hash) vs. constraints required by MAGIC (green), OSCAR NOR (yellow), and OSCAR OR (red).

To further illustrate that OSCAR is significantly more compatible with real resistive devices than MAGIC/FELIX, we perform a threshold voltage variation experiment. Based on our SPICE model, we introduce normal distributions on the switching thresholds of ReRAM cells and perform a Monte Carlo analysis [17] on MAGIC NOR and OSCAR NOR. Figure 12 shows the yield of OSCAR and MAGIC versus four levels of variation from the mean threshold voltage μ (2 V) inside one standard deviation σ. We define yield based on if the output cell stores the correct value after the NOR operation. We run 1000 samples, where each sample contains the minimum circuit needed to perform NOR (e.g., for MAGIC: *Input A*, *Input B*, *Output* in crossbar topology, with the inputs randomly initialized). For OSCAR NOR, the ReRAM cells can only be set when the primitive executes, and thus only V_{set} variation impacts the final yield. In MAGIC, the cells can be set or reset due to the need for output initialization, so the yield for MAGIC is subject to variation for both V_{set} and V_{reset}, which we impose to have the same normal distribution in our experiments. We observe that OSCAR NOR achieves better yield at all four tested variation levels compared to MAGIC NOR, with the most significant yield improvement of 19% at 0.3μ variation.

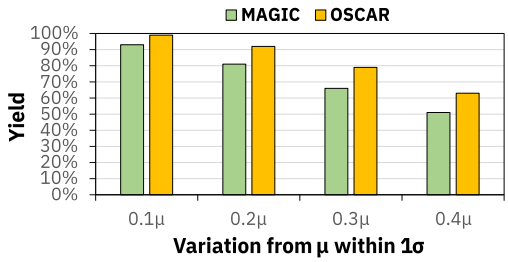


Fig. 12: Yield of OSCAR NOR vs. MAGIC NOR, with different 1σ variations.

We observe two types of failure that occur in OSCAR NOR: (1) output set failure (less than 10% for all four variation levels); and (2) input/output overwriting (more than 90% for all four variation levels). Set failure happens for the NOR(0,0) = 1 case, where the output cell has a high set threshold due to variation, and fails to switch. We find that this type of failure is rare if Δ is set to 75% V_{set} (1.5 V), which provides the output with enough additional assertion voltage to switch. Input overwriting happens more frequently, specifically when the set threshold is higher for the output and lower for the inputs compared to V_{set}. This effectively counteracts the additional voltage Δ that we apply to the output, as the inputs can switch at a lower voltage threshold. Output overwriting happens for NOR(0,1), NOR(1,0) and NOR(1,1), during which the output should not change from its initialized value but does. In these cases, the voltage drop on the output cell is around Δ, which can make output cells with a very low set threshold accidentally switch.

D. Role of Access Topology

While much of the literature on resistive-memory-based PUM focuses on using the 1S1R topology, this topology is not a requirement, and we design RACER to work with either 1S1R or 1T1R. There are several aspects of the performance, area consumption, and energy usage that can be considered when determining the merits of each access topology.

As a driving example, we study how OSCAR works in both access topologies. We speculate that MAGIC and FELIX can also be extended to 1T1R without major changes in the switching threshold constraints. Figure 13 shows the assertion voltages required to perform OSCAR OR in 1S1R and 1T1R arrays.

A key consideration in using the 1T1R topology is whether the voltage between the access transistor's gate (asserted with V_{sel}) and the row line (asserted with V_{float}) exceeds the threshold voltage of the transistor. In principle, this can be

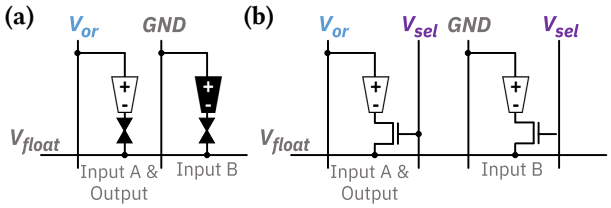


Fig. 13: (a) Voltage assertions for 1S1R OSCAR OR; (b) voltage assertions for 1T1R OSCAR OR.

accomplished with a high enough value of V_{set}, though the value required could in some cases exceed the maximum voltage available from the CMOS drivers. At a high level, the basic functionalities of MAGIC, FELIX, and OSCAR are all compatible with 1T1R topologies, though a more detailed analysis using a particular select transistor technology is necessary.

From an area perspective, there are two considerations. First, while BEOL integration allows for the control and peripheral circuitry to fit *under* a 1S1R crossbar array, this circuitry must be placed *around* the perimeter of a 1T1R array. For the 1T1R array, the CMOS within the array area is fully dedicated to select transistors. Second, the 1T1R cell size is dictated by the area of the access transistor, which is at least $6F^2$ (where F is technology feature size), while the 1S1R cell size can be as small as $4F^2$ For 1S1R, this is because a resistive device can fit underneath the cross section of the minimum-width column and row lines, which are typically $2F$ wide. The 1T1R cell also requires more wires (see Figure 13), which may further increase the cell size. Thus, implementing RACER (and likely other PUM architectures) with 1T1R has a higher area cost than with 1S1R. Section VII-B provides a quantitative discussion of the area costs using 1T1R and using 1S1R.

From an energy perspective, the 1T1R topology requires an additional set of metal wires to control the access transistors, which increases the total energy consumption due to extra wiring load. This penalty is likely to be rather modest, because the resistive cell write energy is typically much larger than charging the gate capacitance and the connecting wires. The energy consumption of the resistive selector in 1S1R and the access transistor in 1T1R depends on the specific device implementation. However, we speculate that the resistive selector and access transistor can consume a similar amount of energy by controlling their fabrication parameters (e.g., transistor width, selector material).

While the above analysis suggests that the 1T1R topology is likely to suffer in comparison to 1S1R, especially in terms of area, there may be some important opportunities in the topology as well. For example, if one allows for different select voltages, there is the possibility of controlling how much voltage drops across the selector, which could make the logic primitive switching more robust against variation. A clear use case for this scenario is under examination.

V. Interface Circuits for RACER

We design RACER not only to deliver high performance and energy savings while using small memory tiles, but also to provide an interface to underlying devices that includes: (1) *decode & drive units*, and (2) *I/O circuitry*. These components help integrate RACER with different resistive memory technologies without changing the bulk of the technology-agnostic control logic. Further, as all instructions in the RACER ISA can be decomposed into micro-ops, these micro-ops can be modified depending on the underlying technology without changing the ISA. In this section, we describe in detail the micro-op and interface circuits that allow RACER to adapt to MAGIC, FELIX, and OSCAR.

A. Decode & Drive Unit Circuitry

When a micro-op reaches the head of the micro-op queue, it is dispatched to a *decode & drive* unit that consumes the micro-op. Figure 14 shows the fields held within the micro-op, and how these fields drive the different components that make up the decode & drive unit. As shown in Figure 14a, a micro-op consists of (1) a $log_2(p)$-bit *Opcode* field to encode p different primitives, (2) a 2-bit buffer selection field (*Buff. Sel.*) that determines if one of the adjacent buffers should be connected to the tile (i.e., if the micro-op is for an inter-tile copy, with the two bits indicating whether to copy to the left, copy to the right, or not copy), and (3) three 6-bit fields (i.e., inputs *Col. A* and *Col. B*, and output *Col. C*) to indicate the input and output columns for a primitive (Figure 14a).

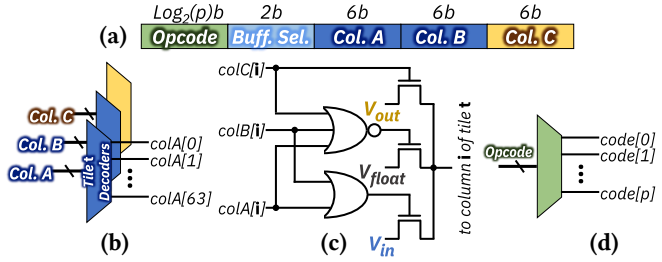


Fig. 14: (a) RACER's micro-op fields; (b) column decoders; (c) column driver (driver passgates are shown as NMOS transistors for simplicity); (d) opcode decoder.

In general, for all resistive memory technologies that we evaluate, one of three voltages is asserted to each column of a tile in every cycle. Figures 14b and 14c show two lightweight circuits used to select which of a set of voltages is asserted on each column in a tile. Each of the 6-bit fields from the micro-op are used to drive a *column decoder*, which sets exactly one of its outputs as shown in Figure 14b. If the column decoder output $colC[i]$ is high, then column i in the tile is the output, and our *column driver* circuit (Figure 14c) asserts V_{out} for that column. If either $colA[i]$ or $colB[i]$ is high, then column i is an input and V_{in} is asserted. Otherwise, column i is not needed for the current micro-op instruction, and V_{float} is asserted. Depending on which primitive is being dispatched in the current cycle, the values of V_{in}, V_{out}, and V_{float} can change (e.g., performing the NAND primitive from FELIX [23] requires V_{in} to equal 0.67× the voltage used when performing NOR). Thus, for devices that support multiple logic primitives, an opcode decoder is required to select the correct primitive (Figure 14d) and drive V_{in}, V_{out}, and V_{float}

from the appropriate voltage rails. For single-primitive logic families such as MAGIC [33], it is not necessary to have an opcode field, and thus no opcode decoding is required. For logic families with destructive primitives, the column decoders and drivers can be further simplified because only two column addresses are required for a given micro-op.

B. Read/Write Circuitry

We build lightweight circuitry to read data out of and write data into tiles, designing this circuitry carefully to avoid diminishing the memory density. Figure 15a shows our circuitry for reading data from a ReRAM cell. We use a voltage divider (consisting of the cell to be read and a reference ReRAM cell) and a skewed inverter to determine if the cell holds logic 0 or logic 1, by detecting the voltage V_{sense}. The reference ReRAM cell is fixed to a low resistance state (logic 1), so when a read voltage V_{read} is applied to the cell to be read, V_{sense} will be one of two values (as shown in Figure 15b): ❶ $V_{sense} \approx 0.5V_{read}$, if the read cell has low resistance (representing logic 1), as there is an equal voltage drop over the read cell and the reference cell; or ❷ $V_{sense} \approx 0\,\text{V}$, if the cell has high resistance (representing logic 0), as most of the voltage drops over the read cell. Because the reference cells can be made using the same resistive material as the memory arrays, this sensing scheme can work with many resistive memory technologies.

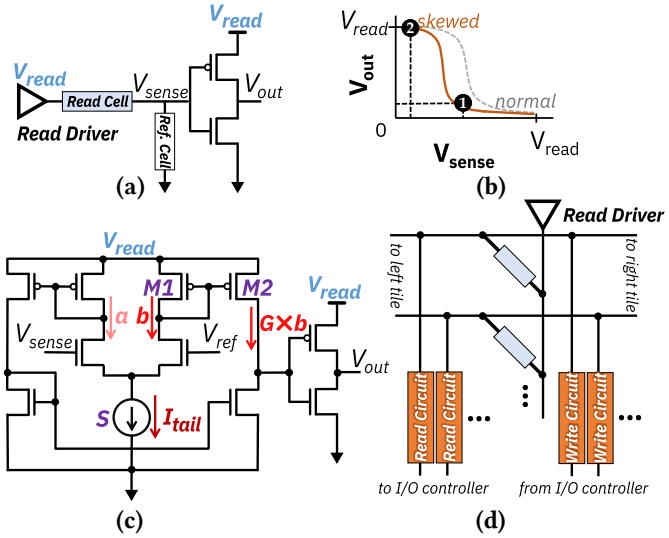


Fig. 15: (a) Read circuitry for one cell; (b) voltage transfer characteristics graph of the skewed inverter used in the read circuitry; (c) generalized read circuitry for tight V_t window; (d) read/write circuitry for a buffer.

Handling Threshold Voltage Compatibility. One issue with the voltage divider based scheme is its response to different threshold voltage values of specific CMOS technologies. If the threshold voltage value causes V_{sense} in Figure 15b to fall outside of the inverter's tolerable input range, we can adjust the reference cell's resistance to any value between logic 1 and logic 0 through geometry or initialization.

To make our sensing scheme more compatible with CMOS technologies that have a tight threshold voltage (V_t) window, we can add a current-mirroring operational transconductance amplifier between the voltage divider and the inverter. Without loss of generality, Figure 15c shows the circuit of this amplifier as designed for $V_{read} > V_{sense} > 0.5V_{read}$. (The NMOS and PMOS transistors can be swapped in cases where a different sensing scheme with $0.5V_{read} > V_{sense} > 0$ is required.) The current source S generates a fixed tail current $I_{tail} = a + b$,[4] where a and b are the current contributions from the left and right current mirrors, respectively. Because a is controlled by V_{sense}, and because V_{ref} is fixed, b is also controlled by V_{sense} (i.e., $b = I_{tail} - a$). b is then mirrored and amplified to $G \times b$, where G is the current gain (which depends on the $M2/M1$ transistor ratio). Depending on V_{sense}'s voltage level, $G \times b$ changes, thereby changing V_{out}. With this operational transconductance amplifier, *any* value of V_{sense} can be used (i.e., the reference cell can be fixed to logic 1), as V_{ref} can be adjusted to ensure correct logical operation.

Using Buffers for I/O. If we were to connect the read circuit to a tile, the voltage divider would not work efficiently, as it would detect the full current of the cell being read as well as partial currents from other cells on the wire. To avoid this scenario, we perform reads only on buffers. Each cell in a buffer is connected to its own read circuit, allowing us to potentially read out the entire contents of a buffer in one cycle.

For the write circuitry, we also cannot write directly to a column in a tile without adding extra peripheral circuitry. Without this additional circuitry, a write with the current decoder design would populate new data into *every* column of a tile. This additional peripheral circuitry would require significant area, diminishing the memory density. Instead, similar to the read circuits, we attach our write circuits only to buffer cells. The write circuit contains a write driver, which asserts a voltage that changes the buffer's resistance. An entire column of data can be written into the buffer in two cycles: the first cycle presets the entire buffer to all zeros, and the second cycle writes logic 1 to only the enabled cells.

Figure 15d shows the buffer-level read/write circuit layout. As we stamp out multiple clusters next to each other in a grid within a chip, we design an I/O controller that provides non-uniform memory access (NUMA) to the other clusters in the chip. The buffer-level read/write circuits are connected to a cluster-level multiplexer, which connects all of the buffers from one pipeline to the I/O controller. At the chip level, each I/O controller connects four neighboring clusters together, enabling local communication across the clusters (and across different pipelines in the same cluster). A controller can burst up to eight cache lines of data (i.e., 512 B) to the CPU on this bus. Our circuit simulations show that moving 512 B of data from an I/O controller to the CPU interface takes 16 ns, and can support a peak bandwidth of 32 GB/s.

VI. Methodology

Modeling & Simulation. We model RACER at the device, circuit, and architecture levels. We develop a highly-detailed Verilog-A model of an ReRAM cell partially based on the

[4]The current source can be turned off when the R/W circuitry is not in use, to reduce the operating power.

aggressively-scaled device parameters in Table 1 of our MICRO 2021 paper [52], while still enforcing the current carrying limit of the wire as discussed in Section III (1 ns switching latency for the resistive element, 1 : 500 ratio of on–off resistances, 0.0128 pJ per switch transition). We synthesize RACER's CMOS-domain components using Synopsis Design Compiler with FreePDK 15 nm [8] to estimate the area, power, and critical path latency of each component, along with delay and power models for the wires connecting these components together (inserting signal repeaters for long wires when necessary). To evaluate microbenchmarks, we use RACER-Sim [6], our detailed open-source simulator for RACER that incorporates the data collected from our Verilog-A model and synthesized circuits. RACER-Sim faithfully models (1) execution at a cell granularity, and (2) all data movement and communication across the data sharing network.

Microbenchmarks. We evaluate RACER using 13 data-intensive microbenchmarks, which span multiple application domains: (1) image processing (*brightness*, *rgb2gray*), (2) linear algebra (*mmul*, *mvmul*, *pagerank*), (3) signal processing (*dftSparse*, *dftDense*), (4) classification (*manhattan*, *hamming*, *lenet5*), and (5) string matching (*grep*, *exactMatch*, *fuzzyMatch*). Due to RACER's custom ISA, we hand-compile microbenchmarks to optimize the data mapping for maximum locality.

VII. Evaluation

We evaluate four different configurations for RACER, each with 4096 clusters: (1) **MAGIC NOR**, a RACER chip that uses MAGIC [33] on 1S1R ReRAM; (2) **MAGIC NAND**, a RACER chip with the same constraints as MAGIC NOR, but using the NAND logic primitive; (3) **FELIX**, a RACER chip that uses FELIX [23] on 1S1R ReRAM;[5] and (4) **OSCAR**, a RACER chip that uses our OSCAR logic family.[6] We scale all configurations to use the same resistive devices with the same on/off resistances and the same switching latency, while allowing the assertion voltages (i.e., V_{in}, V_{out}, V_{float}) to change depending on the requirements of each logic primitive.

A. RACER vs. State-of-the-Art Computing Platforms

Figure 16 summarizes the iso-area performance and energy consumption of the MAGIC NOR configuration of RACER compared to three state-of-the-art platforms, which we discuss in detail in our prior work [52]: (1) **Baseline**, a 16-core CPU modeled after the Intel Xeon Platinum 8253 [29], which uses a conventional off-chip DRAM for main memory; (2) **eMRAM**, a variant of Baseline that replaces the off-chip DRAM with a high-bandwidth (333.3 GB/s for reads) connection to *on-chip* embedded magnetic RAM (MRAM) [15, 53]; and (3) **RTX 2070**, an NVIDIA GPU with 2304 shader cores [47]. On average, RACER provides 107×, 71×, and 12× the performance of Baseline, eMRAM, and RTX 2070, respectively, with energy savings of 189×, 94×, and 17×.

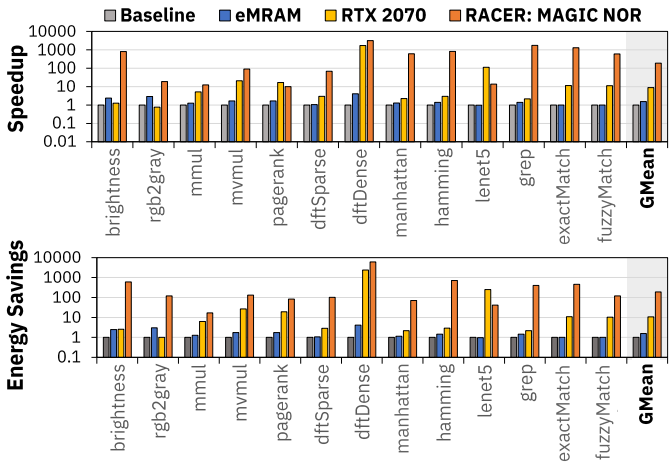


Fig. 16: RACER's speedup (top) and energy savings (bottom) compared to state-of-the-art computing platforms.

B. Area & Circuit Synthesis Analysis

Table I shows a breakdown of the area, static power, and dynamic power consumed by each component of the RACER circuitry. For each cluster, one pipeline control circuit and 64 decode & drive units (one per bit position) are required. The selector passgates are necessary to activate only one pipeline per cluster. Because there are 64 pipelines, each with 64 tiles, and each tile has 64 columns, each requiring 2 passgates, a total of $64 \times 64 \times 64 \times 2$ passgates are required. A similar analysis can be applied to calculate the number of driver passgates (i.e., those that are driving V_{in}, V_{out}, V_{float}). Because the read/write circuitry of a cluster is amortized across all 64 pipelines in the cluster, only 64×64 of them are required per tile (i.e., 64 instances per column of buffers). For 1S1R-based configurations, we use *back-end-of-line* (BEOL) integration, where the ReRAM cell materials can be deposited on metal layers 3–5, with the remaining layers available for building traditional CMOS transistors. Each ReRAM cell occupies a $4\,F^2$ area in a crossbar topology, and we design a cluster such that the ReRAM array, all cluster control, and peripheral circuitry fit within approximately the same footprint to improve area efficiency. Throughout the circuit design process and construction of the RACER chips, we conservatively enforce a maximum power budget of 80 W. From our circuit synthesis, we find that RACER's circuit critical path is 2.6 ns, including wire delay, with each cluster dissipating 0.8 mW of static power. To include a modest guardband, we conservatively clock RACER at 333 MHz (i.e., a 3 ns cycle time). We make two observations from the table.

First, the cluster area for FELIX and for OSCAR is larger than that of MAGIC NOR. This is because both FELIX and OSCAR support multiple logic primitives, which requires wider micro-op queues and opcode decoders. As a result, the cluster area increases by 14% for FELIX and by 11% for OSCAR, compared to MAGIC NOR. We note that despite the

[5]We restrict the FELIX configuration to only 2-input primitives (i.e., 2-input NAND, NOR, OR), although support for three or more inputs is possible at the cost of extending the micro-op bit fields and incorporating additional decoders.

[6]The OSCAR configuration can be realized using either the 1S1R or 1T1R topology. The two topologies yield the same speedup and energy savings, but with different area costs. Section VII-B discusses the impact of topologies on area in more details.

TABLE I: Costs of RACER cluster components and global corner-to-corner (C2C) wiring with repeaters.

Component Name	Area (μm^2)	Delay (ps)	Power (μW) Static	Power (μW) Dynamic	# of Instances Per Cluster
Tile C2C Wiring	—	19	0	482	—
Cluster C2C Wiring	—	140	0	714	—
Chip C2C Wiring	—	16×10^3	0	17×10^3	—
1 64×64 ReRAM Tile	3.7	—	—	—	4096
64 Tiles & 64 Buffers	240	—	—	—	64
Pipeline Control Circuit	74×10^3	100	677	775	1
Decode & Drive Unit	277	27	3.11	126	64
Pipeline Selector	76	4	0.54	73	1
Selector Passgates	0.001	0	0	2×10^{-5}	$64\times2\times64\times64$
I/O Controller	9600	174	128	8×10^3	1
Driver Passgate	0.001	0	0	2×10^{-5}	$64\times3\times64\times64$
R/W Circuit	0.001	1	0	1×10^{-5}	64×64
Cluster Totals					
MAGIC NOR (1S1R)	3.4×10^8	—	33×10^5	306×10^5	—
MAGIC NAND (1S1R)	3.4×10^8	—	33×10^5	304×10^5	—
FELIX (1S1R)	3.9×10^8	—	38×10^5	362×10^5	—
OSCAR (1S1R)	3.8×10^8	—	38×10^5	298×10^5	—
OSCAR (1T1R)	5.2×10^8	—	38×10^5	298×10^5	—

increased area, 4096 clusters can still fit within a $4\,cm^2$ chip for both FELIX and OSCAR (assuming a 1S1R topology).

Second, for 1T1R topologies, the access transistor is the limiting factor for memory density. Each transistor occupies a $6\,F^2$ area, and the required control, peripheral, and I/O circuits cannot be overlapped with the memory arrays to save area. For 1T1R, we move RACER's circuit components to the pitch of the memory arrays, increasing the area of each cluster. Despite the area, the RACER chip can still be clocked at 333 MHz using the 1T1R topology. We further generalize the switching energy of 1T1R access transistors to be the same as the switching energy of the 1S1R selectors. Specific 1T1R/1S1R technologies may size their access devices to obtain a desired switching energy, in which case the power consumption of 1T1R vs. 1S1R may differ. Table I highlights the area cost of 1T1R vs. 1S1R. In total, the 1T1R topology increases the chip area of OSCAR by 31% compared to the 1S1R topology. In an iso-area comparison, OSCAR with 1T1R will achieve smaller speedups and energy savings compared to OSCAR with 1S1R, as the maximum number of pipelines/cores it can have active at any given time decreases by 24%.

C. Performance Analysis

Figure 17 shows the performance of RACER for the four logic families that we evaluate. The speedups are normalized to MAGIC NOR. We make five observations from the figure.

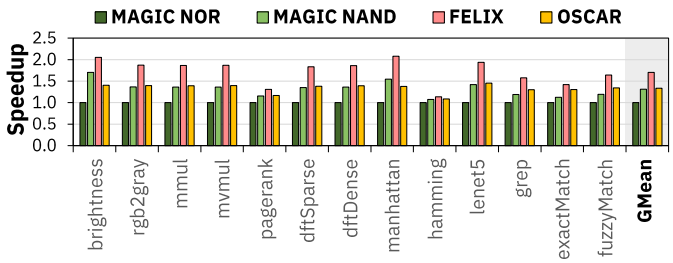


Fig. 17: Performance of different logic families, normalized to MAGIC NOR.

First, MAGIC NAND achieves an average speedup of 31% compared to MAGIC NOR. Because we already restrict the switching energy and latency of MAGIC NAND and MAGIC NOR to be the same, this suggests that logic families that support the NAND (and/or AND) primitive have an advantage over those that do not when integrated with RACER. To understand this better, we break down each microbenchmark into *basic operations* (i.e., NOT, NOR, NAND, OR, COPY, AND, MUX, XOR), as shown in Figure 18a. Each operation can be implemented by each logic family using one or more primitives. The figure shows that many microbenchmarks (e.g., *brightness*, *rgb2gray*, *mmul*) perform a significant number of AND operations because they frequently use multiply or multiply–accumulate instructions. These instructions employ several AND operations to form the partial products. Because an AND operation can be executed in two cycles by MAGIC NAND (compared to five cycles for MAGIC NOR), we conclude that microbenchmarks with frequent multiplications perform better if AND/NAND primitives are available. For microbenchmarks that do not include a significant number of AND operations (e.g., *hamming*), the performance of MAGIC NOR and MAGIC NAND is comparable.

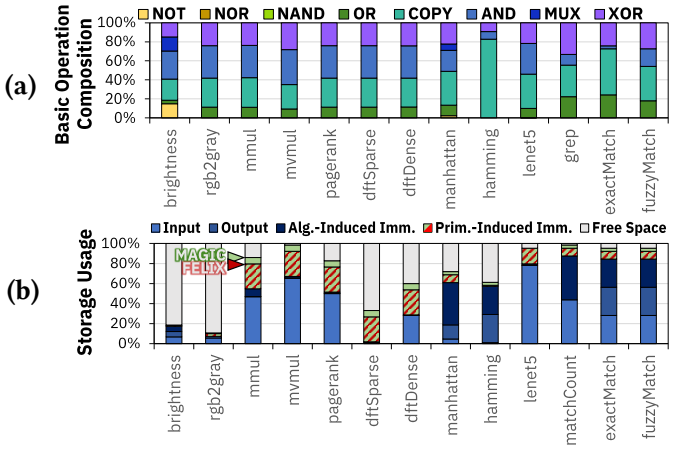


Fig. 18: Breakdown of (a) basic operations and (b) storage usage for our microbenchmarks.

Second, FELIX outperforms MAGIC NOR and MAGIC NAND, by an average of 70% and 23%, respectively, because it supports multiple primitives, which significantly reduces the number of micro-ops needed for execution. In particular, the XOR operation is commonly used by many microbenchmarks due to its frequent use in the addition instruction, which requires two XORs per addition. FELIX can execute XOR in two cycles, as compared to five cycles for MAGIC NOR.

Third, FELIX reduces the number of *primitive-induced intermediates* (i.e., intermediate data generated because a basic operation requires more than one primitive) compared to MAGIC NOR. These intermediates are required because not every basic operation is a primitive (e.g., MAGIC NOR requires one intermediate value when performing an OR operation using primitive NORs). Figure 18b shows the amount of storage required to store primitive-induced intermediates for MAGIC (NOR) in light green, and for FELIX in red. Different logic families do not affect other types of storage (i.e., input, output, and *algorithm-induced intermediates*, which are intermediates created by a microbenchmark's algorithm). Across all microbenchmarks, FELIX reduces the number of primitive-

induced intermediates by an average of 26% compared to MAGIC NOR, thanks to its multi-primitive support.

Fourth, OSCAR achieves a speedup of 33% compared to MAGIC NOR. OSCAR's inclusion of an OR primitive helps to reduce the latencies of multiple basic operations (e.g., NAND, XOR, MUX) over MAGIC NOR, which needs an additional cycle to realize these basic operations. We note that the performance of OSCAR remains the same regardless of topology in these results, as we evaluate both configurations with 4096 clusters (even though OSCAR with 1T1R consumes more area than OSCAR with 1S1R).

Fifth, although the OR primitive is destructive for OSCAR and does not require an output initialization step (unlike the other logic families), these savings do not contribute notably to OSCAR's speedup over MAGIC NOR. In RACER, the output initialization step (e.g., presetting the output column to logic 1 in MAGIC NOR) and logic assertion step (e.g., asserting a voltage on the output column of $V_{out} = V_{nor}$ in MAGIC NOR) are done in a single cycle. Because primitives across most resistive memory technologies require output initialization prior to logic assertion, RACER can couple these two steps and execute them one after another in one cycle. In this case, destructive OR remains idle during the output initialization step. To exploit the fact that destructive primitives do not require output initialization, we can potentially split the two steps, execute them in two cycles, and double the clock frequency. However, doing so doubles the number of micro-ops stored in the micro-op queues, as all non-destructive primitives would now require two micro-ops (one for output initialization, and another for logic assertion) to complete.

We conclude that RACER can integrate with many logic families that support different types of primitives. Further, we conclude that OSCAR can be efficiently integrated with RACER, providing notable performance and energy improvements over MAGIC NOR while enabling compatibility with typical ReRAM devices.

D. Energy & Power Analysis

Figure 19 shows the energy savings of RACER with different logic families. The energy savings are normalized to RACER with MAGIC NOR. We make two observations from the figure.

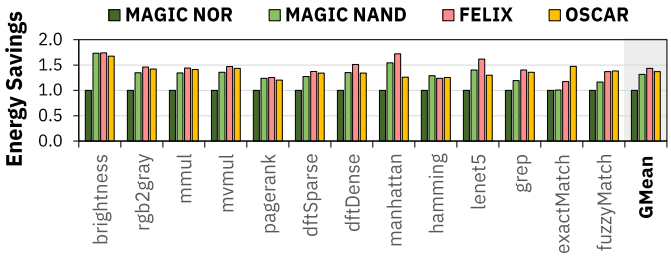


Fig. 19: Energy savings for logic families, normalized to MAGIC NOR.

First, the energy savings of MAGIC NAND and FELIX track with their speedups, with an average savings of 32% and 43%, respectively, compared to MAGIC NOR. This is because both configurations reduce the number of primitives required to perform basic operations thanks to their support of the NAND primitive (and AND, in the case of FELIX). Having fewer primitives executed per basic operation not only reduces latency but also energy, as each additional primitive executed adds additional energy due to resistive switching.

Second, OSCAR provides significant energy savings compared to MAGIC NOR, with an average savings of 37%. Although the configurations lack supports for NAND/AND primitives, OSCAR's primitives use less energy than all of the other technologies that we evaluate. Because OSCAR's OR primitive is destructive, it does not consume energy during the output initialization step, and OSCAR's NOR primitives save 1.83× and 3.35× the energy compared to MAGIC's NOR primitive, due to the higher combined total resistance of the cells involved in OSCAR NOR execution. This allows OSCAR's basic operations to save significant energy compared to the same operations in MAGIC NOR.

While the inclusion of multiple primitives in a logic family can reduce energy usage, it can also increase the static power consumption. Figure 20 shows three properties of a RACER chip that are affected by the number of primitives: area, static power, and the total number of clusters that can fit in a $4\,cm^2$ chip. We observe that OSCAR and FELIX consume more power in order to support two and three primitives, respectively, than single-primitive MAGIC NOR and MAGIC NAND. With three primitives, the micro-op's *Opcode* field needs two bits, requiring larger micro-op queues and opcode decoders to identify the right primitive. This increases the total static power consumption by 15% for FELIX compared to the MAGIC configurations.

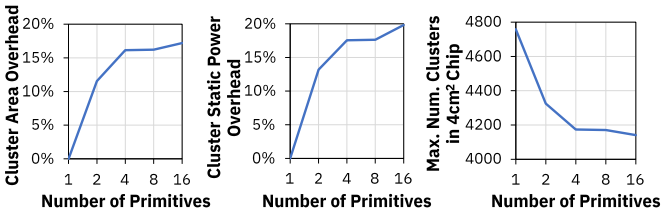


Fig. 20: Area (left) and static power (center) overheads for supporting multiple primitives in 1S1R, normalized to single-primitive 1S1R; maximum number of 1S1R clusters that can fit in a $4\,cm^2$ chip (right).

We conclude that RACER's control circuitry and decode & drive units have low overheads when supporting technologies with multiple primitives. Further, we conclude that OSCAR achieves almost the same energy savings as the theoretically best technology (FELIX), while being significantly more feasible to implement due to its relaxed switching constraints.

E. Comparison to CASCADE

To demonstrate RACER's support for ML applications that require high performance (e.g., real-time neural network inferencing), we compare MAGIC NOR, MAGIC NAND, FELIX, and OSCAR to an area-equivalent state-of-the-art CASCADE [13] accelerator, which uses ReRAM to perform multi-bit analog dot products for neural networks. CASCADE does not support most of RACER's instructions, and has

a fixed neural-network-centric dataflow, so we can compare performance only using the *mmul*, *mvmul*, *dftSparse*, *dftDense*, and *lenet5* microbenchmarks (the only five of our benchmarks that CASCADE can execute). Because these applications heavily rely on the multiply–accumulate (MAC) instruction, and because we are comparing against an accelerator whose primary objective is high performance, we design a new MAC instruction for RACER that reduces the instruction latency by 4.34× at an increased energy cost of 1.7×, compared to the MAC instruction we use in the microbenchmarks presented in Section VII-A and in our MICRO 2021 paper [52]. Thanks to RACER's micro-op interface, its ISA abstraction, and its ability to perform any combinational logic inside memory arrays, creating a new instruction is only a matter of providing the control logic with the correct set of logic primitives. No circuit modification is needed to support the new high-performance MAC instruction.

Figure 21 shows the speedup of RACER with different logic families compared to CASCADE for the microbenchmarks that CASCADE can execute. Compared to CASCADE, RACER achieves an average speedup of 2.22×, 3.04×, 4.15×, 3.16×, and 3.16× using the MAGIC NOR, MAGIC NAND, FELIX, and OSCAR logic families, respectively. We observe that while *mmul*, *mvmul*, and *dftDense* are memory-bound on CASCADE due to their large working set sizes, and incur significant data swapping operations with external memory, *dftSparse* and *lenet5* do not require data swapping and fully benefit from the fast analog dot product.

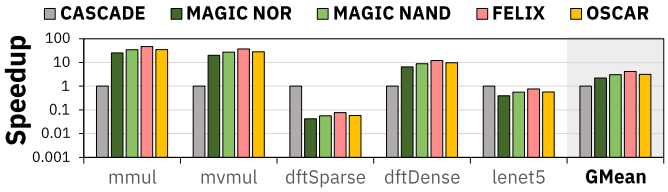


Fig. 21: Speedup vs. CASCADE using low-latency, high-energy MAC instructions

Overall, we find that our new high-performance MAC instruction allows RACER to outperform CASCADE for many, but not all, microbenchmarks. We conclude that RACER trades off some opportunities to accelerate neural networks in order to provide much broader acceleration opportunities, which can be beneficial for mobile and edge applications that build additional non-ML functionality on top of ML model outputs.

VIII. Related Work

To our knowledge, this work is the first to (1) demonstrate that a single PUM architecture can support many different logic families, and (2) propose a logic family that is highly compatible with typical ReRAM devices.

Processing-Using-MRAM. Existing PUM architectures that make use of MRAM devices have been primarily based on 1T1R topologies [2, 3, 4, 38], though architectures with 1S1R topologies are also feasible. In Section VII, we integrate RACER with both 1T1R and 1S1R topologies, showing that the architecture could adapt to MRAM from a topology perspective. From a device perspective, MRAM devices are typically controlled using currents while ReRAM devices are controlled by voltages. The difference in control domain may require modifications to the decode & drive units before RACER can support MRAM, but the control logic can remain unchanged. For example, a recent work examines how 1T1R MRAM can support the MAGIC logic family [42]. The work manages the low resistance contrast of MRAM devices by adjusting access transistor gate voltages.

Analog Processing-Using-ReRAM. Crossbars enable the ability to perform multiple dot products in parallel using ReRAM, by mapping (1) the input matrix values to multi-bit ReRAM cells, and (2) the input vector to analog voltages applied on every column. PUM architectures that use this primitive (e.g., [5, 13, 50, 51]) typically target neural network inference. However, they sacrifice memory density because they require: (1) area-intensive digital/analog converters to perform analog dot products, and (2) dedicated CMOS logic near ReRAM tiles to reduce the partial results generated in the tile. Moreover, multi-bit analog operations are difficult to perform reliably in ReRAM, because of device non-linearity that requires significant precision to discern between adjacent bit representations. In contrast, RACER uses an all-digital approach that avoids the need for costly supplemental logic components and significantly increases reliability, while offering a much larger set of PUM operations that can handle a wide range of data-intensive applications. We briefly compare RACER to CASCADE [13] in Section VII-E.

Digital Processing-Using-ReRAM. Several PUM architectures demonstrate the ability to perform logic using ReRAM crossbars (e.g., [20, 23, 25, 26, 28, 33, 34, 37, 57, 59]). However, these architectures are often limited in the throughput they can achieve without the assistance of discrete logic elements, due to practical limits on array sizes when performing whole-column operations. RACER's bit-pipelining execution model and controller circuitry provide a way to enable high-throughput, low-energy bit-serial computation using small memory arrays.

IX. Conclusion

RACER is a promising architecture for processing using resistive memory. We demonstrate that while the original RACER work was based on a NOR-capable ReRAM device, several alternative logic families can significantly improve the performance of RACER. We develop a new ReRAM-based logical family called OSCAR, which overcomes the difficult-to-achieve constraints on ReRAM switching voltages that prior ReRAM-based logic families suffer from. Overall, we demonstrate how alternative logic families and access topologies can be integrated into RACER with only limited circuit-level modifications, and how several of these approaches can improve the performance and energy consumption of RACER. We conclude that RACER can be an effective component in mobile and edge platforms, which need to integrate efficient machine learning (ML) computation with the ability to perform a range of non-ML operations.

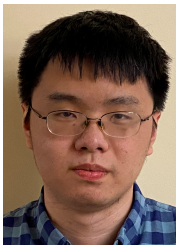
ACKNOWLEDGMENTS

This work was funded in part by a seed grant from the Wilton E. Scott Institute for Energy Innovation, and by the Data Storage Systems Center at Carnegie Mellon University. Minh S. Q. Truong is supported by an Apple Ph.D. Fellowship in Integrated Systems.

REFERENCES

[1] S. Aga *et al.*, "Compute Caches," in *HPCA*, 2017.
[2] S. Angizi *et al.*, "PIMA-Logic: A Novel Processing-in-Memory Architecture for Highly Flexible and Energy-Efficient Logic Computation," in *DAC*, 2018.
[3] S. Angizi *et al.*, "CMP-PIM: An Energy-Efficient Comparator-based Processing-in-Memory Neural Network Accelerator," in *DAC*, 2018.
[4] S. Angizi *et al.*, "AlignS: A Processing-in-Memory Accelerator for DNA Short Read Alignment Leveraging SOT-MRAM," in *DAC*, 2019.
[5] A. Ankit *et al.*, "PUMA: A Programmable Ultra-Efficient Memristor-Based Accelerator for Machine Learning Inference," in *ASPLOS*, 2019.
[6] ARCANA Research Group, "RACER Artifacts — Zenodo Repository," https://github.com/ARCANA-Research/RACER-Artifacts/, 2021, archived at https://doi.org/10.5281/zenodo.5495803.
[7] K. E. Batcher, "Bit-Serial Parallel Processing Systems," *TC*, May 1982.
[8] K. Bhanushali and W. R. Davis, "FreePDK15: An Open-Source Predictive Process Design Kit for 15nm FinFET Technology," in *ISPD*, 2015.
[9] J. Borghetti *et al.*, "Memristive Switches Enable Stateful Logic Operations via Material Implication," *Nature*, Apr. 2010.
[10] A. Boroumand *et al.*, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," in *ASPLOS*, 2018.
[11] A. Chen, "A Review of Emerging Non-Volatile Memory (NVM) Technologies and Applications," *SSE*, Nov. 2016.
[12] Y. Y. Chen *et al.*, "Understanding of the Endurance Failure in Scaled HfO_2-Based 1T1R RRAM Through Vacancy Mobility Degradation," in *IEDM*, 2012.
[13] T. Chou *et al.*, "CASCADE: Connecting RRAMs to Extend Analog Dataflow in an End-to-End In-Memory Processing Paradigm," in *MICRO*, 2019.
[14] W. J. Dally, "Challenges for Future Computing Systems," keynote talk at HiPEAC, 2015.
[15] X. Dong *et al.*, "Circuit and Microarchitecture Evaluation of 3D Stacking Magnetic RAM (MRAM) as a Universal Memory Replacement," in *DAC*, 2008.
[16] C. Eckert *et al.*, "Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks," in *ISCA*, 2018.
[17] M. A. El-Kady, "Probabilistic Short-Circuit Analysis by Monte Carlo Simulations," *TPAS*, May 1983.
[18] A. Fantini *et al.*, "Intrinsic Switching Variability in HfO_2 RRAM," in *IMW*, 2013.
[19] D. Fujiki *et al.*, "Duality Cache for Data Parallel Acceleration," in *ISCA*, 2019.
[20] P. Gaillardon *et al.*, "The Programmable Logic-in-Memory (PLiM) Computer," in *DATE*, 2016.
[21] F. Gao *et al.*, "ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs," in *MICRO*, 2019.
[22] S. Ghose *et al.*, "Processing-in-Memory: A Workload-Driven Perspective," *IBM JRD*, Nov.–Dec. 2019.
[23] S. Gupta *et al.*, "FELIX: Fast and Energy-Efficient Logic in Memory," in *ICCAD*, 2018.
[24] N. Hajinazar *et al.*, "SIMDRAM: A Framework for Bit-Serial SIMD Processing Using DRAM," in *ASPLOS*, 2021.
[25] S. Hamdioui *et al.*, "Memristor for Computing: Myth or Reality?" in *DATE*, 2017.
[26] S. Hamdioui *et al.*, "Memristor-Based Computation-in-Memory Architecture for Data-Intensive Applications," in *DATE*, 2015.
[27] C. Hsu *et al.*, "Self-Rectifying Bipolar TaO_x/TiO_2 RRAM With Superior Endurance Over 1012 Cycles for 3D High-Density Storage-Class Memory," in *VLSIT*, 2013.
[28] M. Imani *et al.*, "FloatPIM: In-Memory Acceleration of Deep Neural Network Training With High Precision," in *ISCA*, 2018.
[29] Intel Corp., "Intel Xeon Platinum 8253," https://ark.intel.com/content/www/us/en/ark/products/192465/intel-xeon-platinum-8253-processor-22m-cache-2-20-ghz.html.
[30] S. Jekola *et al.*, "A 28 nm Configurable Memory (TCAM/BCAM/SRAM) Using Push-Rule 6T Bit Cell Enabling Logic-in-Memory," in *JSSC*, 2016.
[31] U. Kang *et al.*, "Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling," in *The Memory Forum*, 2014.
[32] G. Kestor *et al.*, "Quantifying the Cost of Data Movement in Scientific Applications," in *IISWC*, 2013.
[33] S. Kvatinsky *et al.*, "MAGIC: Memristor-Aided Logic," *TCAS II*, Sep. 2014.
[34] S. Kvatinsky *et al.*, "Memristor-Based IMPLY Logic Design Procedure," in *ICCD*, 2011.
[35] S. Kvatinsky *et al.*, "Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies," in *VLSI*, 2013.
[36] S. Lee *et al.*, "Multi-Level Switching of Triple-Layered TaO_x RRAM With Excellent Reliability for Storage Class Memory," in *VLSIT*, 2012.
[37] Y. Levy *et al.*, "Logic Operations in Memory Using a Memristive Akers Array," *Microelectronics*, Nov. 2014.
[38] B. Li *et al.*, "Merging the Interface: Power, Area, and Accuracy Co-Optimization for RRAM Crossbar-Based Mixed-Signal Computing System," in *DAC*, 2015.
[39] S. Li *et al.*, "DRISA: A DRAM-Based Reconfigurable In-Situ Accelerator," in *MICRO*, 2017.
[40] S. Li *et al.*, "Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-Volatile Memories," in *DAC*, 2016.
[41] R. Liu *et al.*, "Experimental Characterization of Physical Unclonable Function Based on 1 kb Resistive Random Access Memory Arrays," *EDL*, Oct. 2015.
[42] J. Louis *et al.*, "Performing Memristor-Aided Logic (MAGIC) Using STT-MRAM," in *ICECS*, 2019.
[43] D. K. Maiti *et al.*, "Composition-Dependent Nanoelectronics of Amido-Phenazines: Non-Volatile RRAM and WORM Memory Devices," *Nature*, Oct. 2017.
[44] A. Makarov *et al.*, "Emerging Memory Technologies: Trends, Challenges, and Modeling Methods," *Microelectronics Reliability*, Apr. 2012.
[45] J. A. Mandelman *et al.*, "Challenges and Future Directions for the Scaling of Dynamic Random-Access Memory (DRAM)," *IBM JRD*, Mar. 2002.
[46] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," in *IMW*, 2013.
[47] NVIDIA Corp., "Geforce RTX 2070," https://www.nvidia.com/en-us/geforce/graphics-cards/rtx-2070/.
[48] V. Seshadri *et al.*, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.
[49] V. Seshadri and O. Mutlu, "Simple Operations in Memory to Reduce Data Movement," in *Advances in Computers*. Elsevier, 2017, vol. 106.
[50] A. Shafiee *et al.*, "ISAAC: A Convolutional Neural Network Accelerator With In-Situ Analog Arithmetic in Crossbars," in *ISCA*, 2016.
[51] L. Song *et al.*, "PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning," in *HPCA*, 2017.
[52] M. S. Q. Truong *et al.*, "RACER: Bit-Pipelined Processing Using Resistive Memory," in *MICRO*, 2021.
[53] J. Wang *et al.*, "Enabling High-Performance LPDDRx-Compatible MRAM," in *ISLPED*, 2014.
[54] H. E. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Pearson/Addison-Wesley, 2010.
[55] H.-S. P. Wong *et al.*, "Metal-Oxide RRAM," *Proc. IEEE*, Jun. 2012.
[56] J. Woo *et al.*, "Improved Synaptic Behavior Under Identical Pulses Using AlO_x/HfO_2 Bilayer RRAM Array for Neuromorphic System," *EDL*, Aug. 2016.
[57] L. Xie *et al.*, "Fast Boolean Logic Mapped on Memristor Crossbar," in *ICCD*, 2015.
[58] C. Ye *et al.*, "Enhanced Resistive Switching Performance for Bilayer HfO_2/TiO_2 Resistive Random Access Memory," *SST*, Sep. 2016.
[59] J. Yu *et al.*, "Memristive Devices for Computation-in-Memory," in *DATE*, 2018.
[60] S. Yu and Y. P. Chen, "Emerging Memory Technologies: Recent Trends and Prospects," *SCC-M*, Spring 2016.
[61] S. Yu *et al.*, "Binary Neural Network With 16 Mb RRAM Macro Chip for Classification and Online Training," in *IEDM*, 2016.
[62] Y. Zha and J. Li, "Liquid Silicon: A Data-Centric Reconfigurable Architecture Enabled by RRAM Technology," in *FPGA*, 2018.

Minh S. Q. Truong is a Ph.D. student in the Department of Electrical and Computer Engineering at Carnegie Mellon University. He received dual B.S. degrees in electrical engineering and in computer engineering from the University of California, Davis in 2019. His current Ph.D. research seeks to create new classes of computer systems based on the processing-in-memory paradigm to reduce the power consumption of data-intensive applications by orders of magnitudes, and to enable efficient edge and cloud computing. His general research interest lies at the intersection of computer systems, microarchitecture, circuits, and how to design a holistic computer system.



Liting Shen is a Ph.D. student in the Department of Electrical and Computer Engineering at Carnegie Mellon University. He received his B.S. degree in applied physics from the Hong Kong Polytechnic University in 2017. His research interests lies in making Resistive Random Access Memory (ReRAM) cells to compute. His work focuses on theoretical analysis and experimental demonstration of ReRAM-based logic operations like OR, NOT and NOR.

Alexander Glass is an M.S. student in the Department of Electrical and Computer Engineering at Carnegie Mellon University. He received his B.S. in electrical and computer Engineering from Carnegie Mellon University in 2022. His current research interests include processing-in-memory architectures and CPU design.

Alison Hoffmann is an M.S. student in the Department of Electrical and Computer Engineering at Carnegie Mellon University. She received her B.S.in electrical and computer Engineering from Carnegie Mellon University in 2021. She is broadly interested in computer architecture, processing-using-memory, and data center computer architectures.

L. Richard Carley received an S.B. in 1976, an M.S. in 1978, and a Ph.D. in 1984, all from the Massachusetts Institute of Technology. He joined the Electrical and Computer Engineering Department at Carnegie Mellon University (CMU) in Pittsburgh Pennsylvania in 1984, and in March 2001, he became the STMicroelectronics Professor of Engineering at CMU. Dr. Carley's research interests include analog and RF integrated circuit design in deeply scaled CMOS technologies, and novel micro-electro-mechanical and nano-electro-mechanical device design and fabrication. For the past several years, Dr. Carley has studied the design of efficient RF Power Amplifiers in advanced BiCMOS technologies. Dr. Carley has been granted 27 patents, authored or co-authored over 250 technical papers, and authored or co-authored over 20 books and/or book chapters. He has won numerous awards including Best Technical Paper Awards at both the 1987 and the 2002 Design Automation Conference (DAC), a "Most Influential Paper" award from DAC, and a "Best Panel Session" award at ISSCC in 1993. In 1997, Dr. Carley co-founded the analog electronic design automation startup, Neolinear, which was acquired by Cadence in 2004.

James A. Bain is a professor in the Electrical and Computer Engineering Department of Carnegie Mellon University. He also holds a courtesy appointment in the Department of Materials Science and Engineering and is Associate Director of the Data Storage Systems Center (DSSC). Professor Bain received his B.S. (1988) in materials science and Engineering from the University of Pennsylvania and his M.S. (1991) and Ph.D. (1993), also in materials science and engineering, from Stanford University. He has co-authored more than 300 papers in the field of magnetic, optical, electrical, thermal, and mechanical devices and materials for information technology. He currently has active research programs in heat assisted magnetic recording, and resistive switches for memory and reconfigurable electronics. He is a senior member of IEEE (Societies: Magnetics, Electron Devices, Photonics).

Saugata Ghose is an assistant professor in the Department of Computer Science at the University of Illinois Urbana-Champaign, and holds an affiliate appointment with the Department of Electrical and Computer Engineering. He received dual B.S. degrees in computer science and in computer engineering from Binghamton University, State University of New York (2007), and received M.S. and Ph.D. degrees in electrical and computer engineering from Cornell University (2014). He earned the best paper award from DFRWS-EU in 2017, and was a 2019 Wimmer Faculty Fellow while at Carnegie Mellon University. His current research interests include data-oriented computer architectures and systems, new interfaces between systems software and hardware, energy-efficient memory and storage, and architectures for emerging platforms and domains. He is a member of the IEEE (Computer Society). For more information, please visit his website at https://ghose.cs.illinois.edu/